Homework 10 - CIFAR10 Image Classification with PyTorch

About

The goal of the homework is to train a convolutional neural network on the standard CIFAR10 image classfication dataset.

When solving machine learning tasks using neural networks, one typically starts with a simple network architecture and then improves the network by adding new layers, retraining, adjusting parameters, retraining, etc. We attempt to illustrate this process below with several architecture improvements.

Dev Environment

Working on Google Colab

You may choose to work locally or on Google Colaboratory. You have access to free compute through this service. Colab is recommended since it will be setup correctly and will have access to GPU resources.

1. Visit https://colab.research.google.com/drive (https://colab.research.google.com/drive)
2. Navigate to the **`Upload`** tab, and upload your `HW10.ipynb`
3. Now on the top right corner, under the `Comment` and `Share` options, you should see a `Connect` option. Once you are connected, you will have access to a VM with 12GB RAM, 50 GB disk space and a single GPU. The dropdown menu will allow you to connect to a local runtime as well.

Notes:

- **If you do not have a working setup for Python 3, this is your best bet. It will also save you from heavy installations like `tensorflow` if you don't want to deal with those.**
- ***There is a downside*. You can only use this instance for a single 12-hour stretch, after which your data will be deleted, and you would have redownload all your datasets, any libraries not already on the VM, and regenerate your logs**.

Installing PyTorch and Dependencies

The instructions for installing and setting up PyTorch can be found at https://pytorch.org/get-started/locally/ (https://pytorch.org/get-started/locally/). Make sure you follow the instructions for your machine. For any of the remaining libraries used in this assignment:

- We have provided a `hw8_requirements.txt` file on the homework web page.
- Download this file, and in the same directory you can run `pip3 install -r hw8_requirements.txt` Check that PyTorch installed correctly by running the following:

In [0]:
```
import torch
torch.rand(5, 3)
```

Out[1]:
```
tensor([[0.6626, 0.3950, 0.5138],
        [0.0493, 0.0939, 0.1592],
        [0.8296, 0.1022, 0.8555],
        [0.2304, 0.8264, 0.6811],
        [0.4494, 0.9821, 0.1311]])
```

Part 0 Imports and Basic Setup (5 Points)

First, import the required libraries as follows. The libraries we will use will be the same as those in HW8.

In [0]:
```
import numpy as np
import torch
from torch import nn
from torch import optim

import matplotlib.pyplot as plt
```

GPU Support

Training of large network can take a long time. PyTorch supports GPU with just a small amount of effort.

When creating our networks, we will call `net.to(device)` to tell the network to train on the GPU, if one is available. Note, if the network utilizes the GPU, it is important that any tensors we use with it (such as the data) also reside on the CPU. Thus, a call like `images = images.to(device)` is necessary with any data we want to use with the GPU.

Note: If you can't get access to a GPU, don't worry to much. Since we use very small networks, the difference between CPU and GPU isn't large and in some cases GPU will actually be slower.

In [0]:
```
import torch.cuda as cuda

# Use a GPU, i.e. cuda:0 device if it available.
device = torch.device("cuda:0" if cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

In [0]:

```
#@title Training Code
import time

class Flatten(nn.Module):
  """NN Module that flattens the incoming tensor."""
  def forward(self, input):
    return input.view(input.size(0), -1)

def train(model, train_loader, test_loader, loss_func, opt, num_epochs=10):
  all_training_loss = np.zeros((0,2))
  all_training_acc = np.zeros((0,2))
  all_test_loss = np.zeros((0,2))
  all_test_acc = np.zeros((0,2))

  training_step = 0
  training_loss, training_acc = 2.0, 0.0
  print_every = 1000

  start = time.clock()

  for i in range(num_epochs):
    epoch_start = time.clock()

    model.train()
    for images, labels in train_loader:
      images, labels = images.to(device), labels.to(device)
      opt.zero_grad()

      preds = model(images)
      loss = loss_func(preds, labels)
      loss.backward()
      opt.step()

      training_loss += loss.item()
      training_acc += (torch.argmax(preds, dim=1)==labels).float().mean()

      if training_step % print_every == 0:
        training_loss /= print_every
        training_acc /= print_every

        all_training_loss = np.concatenate((all_training_loss, [[training_step, t
        all_training_acc = np.concatenate((all_training_acc, [[training_step, tra

        print('  Epoch %d @ step %d: Train Loss: %3f, Train Accuracy: %3f' % (
            i, training_step, training_loss, training_acc))
        training_loss, training_acc = 0.0, 0.0

      training_step+=1

    model.eval()
    with torch.no_grad():
      validation_loss, validation_acc = 0.0, 0.0
      count = 0
      for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        output = model(images)
        validation_loss+=loss_func(output,labels)
        validation_acc+=(torch.argmax(output, dim=1) == labels).float().mean()
        count += 1
      validation_loss/=count
      validation_acc/=count
```

```
62
63        all_test_loss = np.concatenate((all_test_loss, [[training_step, validation_
64        all_test_acc = np.concatenate((all_test_acc, [[training_step, validation_ac
65
66        epoch_time = time.clock() - epoch_start
67
68        print('Epoch %d Test Loss: %3f, Test Accuracy: %3f, time: %.1fs' % (
69            i, validation_loss, validation_acc, epoch_time))
70
71    total_time = time.clock() - start
72    print('Final Test Loss: %3f, Test Accuracy: %3f, Total time: %.1fs' % (
73        validation_loss, validation_acc, total_time))
74
75    return {'loss': { 'train': all_training_loss, 'test': all_test_loss },
76            'accuracy': { 'train': all_training_acc, 'test': all_test_acc }}
77
78  def plot_graphs(model_name, metrics):
79    for metric, values in metrics.items():
80      for name, v in values.items():
81        plt.plot(v[:,0], v[:,1], label=name)
82      plt.title(f'{metric} for {model_name}')
83      plt.legend()
84      plt.xlabel("Training Steps")
85      plt.ylabel(metric)
86      plt.show()
87
```

Load the** CIFA-10** dataset and define the transformations. You may also want to print its structure, size, as well as sample a few images to get a sense of how to design the network.

In [0]:
```
1  !mkdir hw10_data
```

In [0]:
```
1  # Download the data.
2  from torchvision import datasets, transforms
3
4  transformations = transforms.Compose(
5      [transforms.ToTensor(),
6       transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])
7  train_set = datasets.CIFAR10(root='hw10_data/', download=True, transform=transfor
8  test_set = datasets.CIFAR10(root='hw10_data', download=True, train=False, transfo
```

```
0it [00:00, ?it/s]

Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz (https://www.c
s.toronto.edu/~kriz/cifar-10-python.tar.gz) to hw10_data/cifar-10-python.tar.gz

100%|██████████| 170409984/170498071 [00:35<00:00, 2101539.96it/s]

Files already downloaded and verified
```

Use `DataLoader` to create a loader for the training set and a loader for the testing set. You can use a `batch_size` of 8 to start, and change it if you wish.

In [0]:
```
from torch.utils.data import DataLoader

batch_size = 8
train_loader = torch.utils.data.DataLoader(train_set, batch_size, shuffle=True, n
test_loader = torch.utils.data.DataLoader(test_set, batch_size, shuffle=True, num

input_shape = np.array(train_set[0][0]).shape
input_dim = input_shape[1]*input_shape[2]*input_shape[0]

```

In [0]:
```
training_epochs = 5
```

Part 1 CIFAR10 with Fully Connected Neural Netowrk (25 Points)

As a warm-up, let's begin by training a two-layer fully connected neural network model on ** CIFAR-10** dataset. You may go back to check HW8 for some basics.

We will give you this code to use as a baseline to compare against your CNN models.

In [0]:

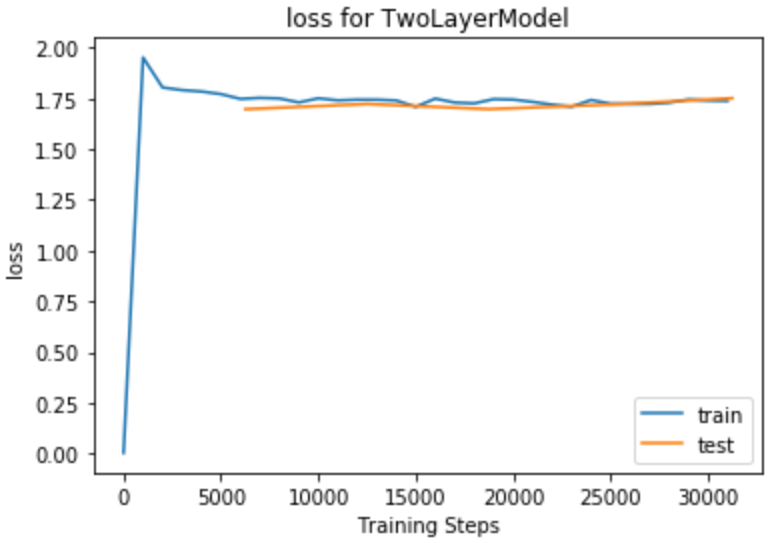
```
class TwoLayerModel(nn.Module):
  def __init__(self):
    super(TwoLayerModel, self).__init__()
    self.net = nn.Sequential(
      Flatten(),
      nn.Linear(input_dim, 64),
      nn.ReLU(),
      nn.Linear(64, 10))

  def forward(self, x):
    return self.net(x)

model = TwoLayerModel().to(device)

loss = nn.CrossEntropyLoss()
optimizer = optim.RMSprop(model.parameters(), lr=0.001, weight_decay=0.01)

# Training epoch should be about 15-20 sec each on GPU.
metrics = train(model, train_loader, test_loader, loss, optimizer, training_epoch
```

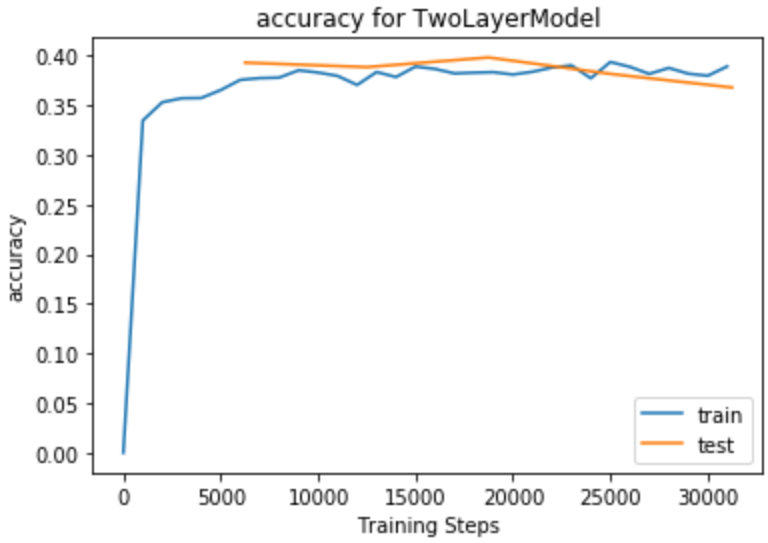
```
  Epoch 0 @ step 0: Train Loss: 0.004303, Train Accuracy: 0.000125

170500096it [00:50, 2101539.96it/s]

  Epoch 0 @ step 1000: Train Loss: 1.951483, Train Accuracy: 0.334500
  Epoch 0 @ step 2000: Train Loss: 1.803716, Train Accuracy: 0.352875
  Epoch 0 @ step 3000: Train Loss: 1.790826, Train Accuracy: 0.356875
  Epoch 0 @ step 4000: Train Loss: 1.783943, Train Accuracy: 0.357125
  Epoch 0 @ step 5000: Train Loss: 1.771074, Train Accuracy: 0.365125
  Epoch 0 @ step 6000: Train Loss: 1.747424, Train Accuracy: 0.375375
Epoch 0 Test Loss: 1.696952, Test Accuracy: 0.392700, time: 17.9s
  Epoch 1 @ step 7000: Train Loss: 1.752460, Train Accuracy: 0.377125
  Epoch 1 @ step 8000: Train Loss: 1.749601, Train Accuracy: 0.377625
  Epoch 1 @ step 9000: Train Loss: 1.730119, Train Accuracy: 0.385000
  Epoch 1 @ step 10000: Train Loss: 1.750363, Train Accuracy: 0.382750
  Epoch 1 @ step 11000: Train Loss: 1.740194, Train Accuracy: 0.379375
  Epoch 1 @ step 12000: Train Loss: 1.743895, Train Accuracy: 0.370250
Epoch 1 Test Loss: 1.722507, Test Accuracy: 0.388200, time: 18.0s
  Epoch 2 @ step 13000: Train Loss: 1.743650, Train Accuracy: 0.383250
  Epoch 2 @ step 14000: Train Loss: 1.739619, Train Accuracy: 0.378250
  Epoch 2 @ step 15000: Train Loss: 1.708270, Train Accuracy: 0.388750
  Epoch 2 @ step 16000: Train Loss: 1.748885, Train Accuracy: 0.386375
  Epoch 2 @ step 17000: Train Loss: 1.729535, Train Accuracy: 0.381875
  Epoch 2 @ step 18000: Train Loss: 1.725448, Train Accuracy: 0.382625
Epoch 2 Test Loss: 1.697185, Test Accuracy: 0.397900, time: 17.4s
  Epoch 3 @ step 19000: Train Loss: 1.746290, Train Accuracy: 0.383125
  Epoch 3 @ step 20000: Train Loss: 1.744132, Train Accuracy: 0.380750
  Epoch 3 @ step 21000: Train Loss: 1.732730, Train Accuracy: 0.383500
  Epoch 3 @ step 22000: Train Loss: 1.719271, Train Accuracy: 0.387750
  Epoch 3 @ step 23000: Train Loss: 1.709115, Train Accuracy: 0.390000
  Epoch 3 @ step 24000: Train Loss: 1.741992, Train Accuracy: 0.377000
Epoch 3 Test Loss: 1.719641, Test Accuracy: 0.381400, time: 17.2s
  Epoch 4 @ step 25000: Train Loss: 1.723885, Train Accuracy: 0.393375
  Epoch 4 @ step 26000: Train Loss: 1.722273, Train Accuracy: 0.388500
  Epoch 4 @ step 27000: Train Loss: 1.722589, Train Accuracy: 0.381375
  Epoch 4 @ step 28000: Train Loss: 1.728848, Train Accuracy: 0.387375
  Epoch 4 @ step 29000: Train Loss: 1.744673, Train Accuracy: 0.381625
  Epoch 4 @ step 30000: Train Loss: 1.739262, Train Accuracy: 0.379500
  Epoch 4 @ step 31000: Train Loss: 1.737691, Train Accuracy: 0.388875
Epoch 4 Test Loss: 1.750870, Test Accuracy: 0.367700, time: 18.0s
Final Test Loss: 1.750870, Test Accuracy: 0.367700, Total time: 88.5s
```

Plot the model results

Normally we would want to use Tensorboard for looking at metrics. However, if colab reset while we are working, we might lose our logs and therefore our metrics. Let's just plot some graphs that will survive across colab instances.

In [0]:
```
1 plot_graphs("TwoLayerModel", metrics)
```





Part 2 Convolutional Neural Network (CNN) (35 Points)

Now, let's design a convolution neural netwrok!

Build a simple CNN model, inserting 2 CNN layers in from of our 2 layer fully connect model from above:

1. A convolution with 3x3 filter, 16 output channels, stride = 1, padding=1
2. A ReLU activation
3. A Max-Pooling layer with 2x2 window
4. A convolution, 3x3 filter, 16 output channels, stride = 1, padding=1
5. A ReLU activation
6. Flatten layer
7. Fully connected linear layer with output size 64
8. ReLU
9. Fully connected linear layer, with output size 10

You will have to figure out the input sizes of the first fully connnected layer based on the previous layer sizes. Note that you also need to fill those in the report section (see report section in the notebook for details)

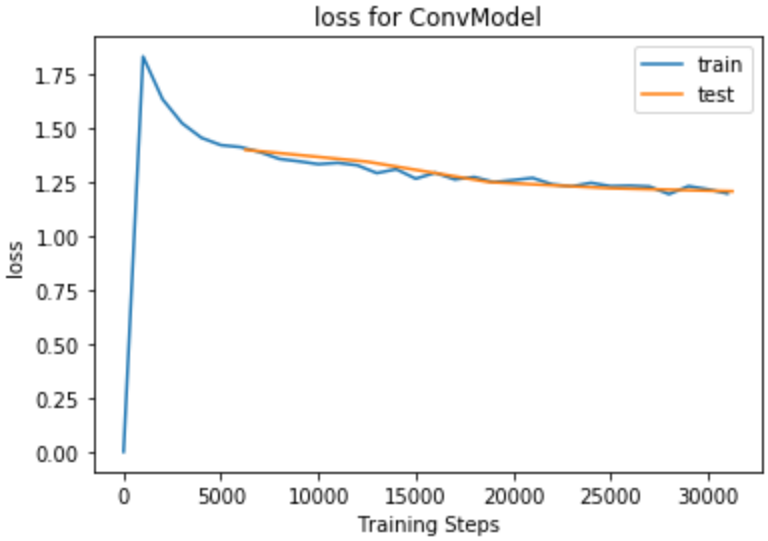
In [0]:

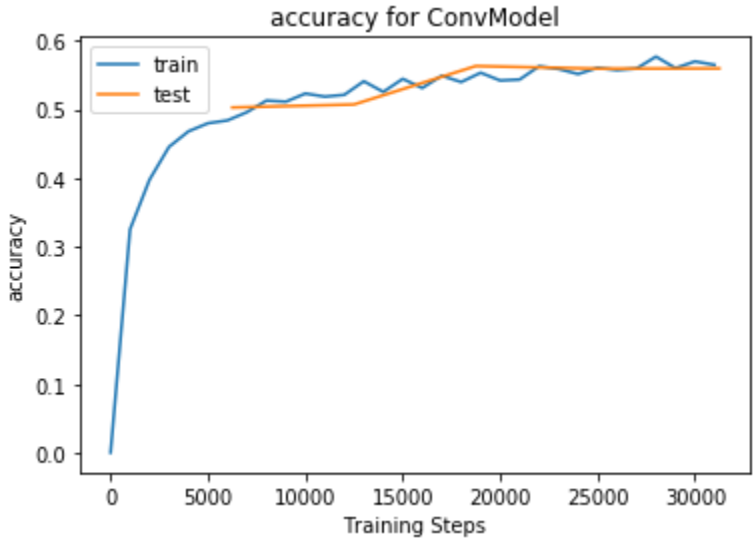
```
import torch.nn.functional as F
class ConvModel(nn.Module):
  # Your Code Here
  def __init__(self):
    super(ConvModel, self).__init__()
    ##define convolution layers
    self.conv1 = nn.Conv2d(in_channels=3,out_channels=16,kernel_size = 3,stride=1
    self.conv2 = nn.Conv2d(in_channels=16,out_channels=16,kernel_size = 3,stride=

    ##define fully connected layers
    self.fc1 = nn.Linear(16*16*16, 64)
    self.fc2 = nn.Linear(64, 10)

    ##define flatten
    self.flatten = Flatten()

  def forward(self, x):
    #Apply convolution1, relu and maxpooling
      x = self.conv1(x)
      x = F.relu(x)
      x = F.max_pool2d(x, (2, 2))##Size([8, 16, 16, 16])

      #Apply convolution2, relu
      x = self.conv2(x)
      x = F.relu(x)

      ##Flatten
      x = self.flatten(x)
      ##fully connected layers
      x = self.fc1(x)
      x = F.relu(x)
      x = self.fc2(x)
      return x

model = ConvModel().to(device)

loss = nn.CrossEntropyLoss()
optimizer = optim.RMSprop(model.parameters(), lr=0.001, weight_decay=0.01)

metrics = train(model, train_loader, test_loader, loss, optimizer, training_epoch
```

```
  Epoch 0 @ step 0: Train Loss: 0.004239, Train Accuracy: 0.000250
  Epoch 0 @ step 1000: Train Loss: 1.832785, Train Accuracy: 0.325375
  Epoch 0 @ step 2000: Train Loss: 1.635091, Train Accuracy: 0.397625
  Epoch 0 @ step 3000: Train Loss: 1.524421, Train Accuracy: 0.445375
  Epoch 0 @ step 4000: Train Loss: 1.456644, Train Accuracy: 0.468000
  Epoch 0 @ step 5000: Train Loss: 1.422762, Train Accuracy: 0.479750
  Epoch 0 @ step 6000: Train Loss: 1.413759, Train Accuracy: 0.483875
Epoch 0 Test Loss: 1.401174, Test Accuracy: 0.502700, time: 24.7s
  Epoch 1 @ step 7000: Train Loss: 1.390499, Train Accuracy: 0.495875
  Epoch 1 @ step 8000: Train Loss: 1.358934, Train Accuracy: 0.512750
  Epoch 1 @ step 9000: Train Loss: 1.347796, Train Accuracy: 0.511125
  Epoch 1 @ step 10000: Train Loss: 1.334809, Train Accuracy: 0.523000
  Epoch 1 @ step 11000: Train Loss: 1.340734, Train Accuracy: 0.518500
  Epoch 1 @ step 12000: Train Loss: 1.329528, Train Accuracy: 0.521125
Epoch 1 Test Loss: 1.346172, Test Accuracy: 0.507100, time: 25.9s
  Epoch 2 @ step 13000: Train Loss: 1.293571, Train Accuracy: 0.540875
  Epoch 2 @ step 14000: Train Loss: 1.311215, Train Accuracy: 0.525375
  Epoch 2 @ step 15000: Train Loss: 1.267555, Train Accuracy: 0.544375
  Epoch 2 @ step 16000: Train Loss: 1.294812, Train Accuracy: 0.531000
  Epoch 2 @ step 17000: Train Loss: 1.264044, Train Accuracy: 0.548875
```

```
  Epoch 2 @ step 18000: Train Loss: 1.275115, Train Accuracy: 0.539750
Epoch 2 Test Loss: 1.251915, Test Accuracy: 0.563000, time: 24.3s
  Epoch 3 @ step 19000: Train Loss: 1.251892, Train Accuracy: 0.553250
  Epoch 3 @ step 20000: Train Loss: 1.261007, Train Accuracy: 0.542000
  Epoch 3 @ step 21000: Train Loss: 1.271045, Train Accuracy: 0.543375
  Epoch 3 @ step 22000: Train Loss: 1.241558, Train Accuracy: 0.562750
  Epoch 3 @ step 23000: Train Loss: 1.231875, Train Accuracy: 0.558750
  Epoch 3 @ step 24000: Train Loss: 1.248294, Train Accuracy: 0.551375
Epoch 3 Test Loss: 1.223390, Test Accuracy: 0.559200, time: 25.9s
  Epoch 4 @ step 25000: Train Loss: 1.232924, Train Accuracy: 0.560375
  Epoch 4 @ step 26000: Train Loss: 1.234988, Train Accuracy: 0.557125
  Epoch 4 @ step 27000: Train Loss: 1.231304, Train Accuracy: 0.559375
  Epoch 4 @ step 28000: Train Loss: 1.195990, Train Accuracy: 0.576750
  Epoch 4 @ step 29000: Train Loss: 1.233033, Train Accuracy: 0.560000
  Epoch 4 @ step 30000: Train Loss: 1.218548, Train Accuracy: 0.569750
  Epoch 4 @ step 31000: Train Loss: 1.198725, Train Accuracy: 0.564750
Epoch 4 Test Loss: 1.209426, Test Accuracy: 0.559500, time: 26.8s
Final Test Loss: 1.209426, Test Accuracy: 0.559500, Total time: 127.7s
```

In [0]:
```
1 plot_graphs("ConvModel", metrics)
```





Do you notice the improvement over the accuracy compared to that in Part 1?

Yes, both accuray and loss imporves compred to part 1.

Part 3 Open Design Competition (35 Points + 10 bonus points)

Try to beat the previous models by adding additional layers, changing parameters, etc. You should add at least one layer.

Possible changes include:

- Dropout
- Batch Normalization
- More layers
- Residual Connections (harder)
- Change layer size
- Pooling layers, stride
- Different optimizer
- Train for longer

Once you have a model you think is great, evaluate it against our hidden test data (see hidden_loader above) and upload the results to the leader board on gradescope. **The top 3 scorers will get a bonus 10 points.**

You can steal model structures found on the internet if you want. The only constraint is that **you must train the model from scratch**.

In [0]:

```
# You Awesome Super Best model code here (ResNet)
class ResidualBlock(nn.Module):
  def __init__(self, inchannel, outchannel, stride=1):
    super(ResidualBlock, self).__init__()
    self.left = nn.Sequential(
        nn.Conv2d(inchannel, outchannel, kernel_size=3, stride=stride, padding=1,
        nn.BatchNorm2d(outchannel),
        nn.ReLU(inplace=True),
        nn.Conv2d(outchannel, outchannel, kernel_size=3, stride=1, padding=1, bia
        nn.BatchNorm2d(outchannel)
    )
    self.shortcut = nn.Sequential()
    if stride != 1 or inchannel != outchannel:
        self.shortcut = nn.Sequential(
            nn.Conv2d(inchannel, outchannel, kernel_size=1, stride=stride, bias=F
            nn.BatchNorm2d(outchannel)
        )
  def forward(self, x):
    out = self.left(x)
    out += self.shortcut(x)
    out = F.relu(out)
    return out
class ResNet(nn.Module):
  def __init__(self, ResidualBlock, num_classes=10):
    super(ResNet, self).__init__()
    self.inchannel = 64
    self.conv1 = nn.Sequential(
        nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False),
        nn.BatchNorm2d(64),
        nn.ReLU(),
    )
    self.layer1 = self.make_layer(ResidualBlock, 64,  2, stride=1)
    self.layer2 = self.make_layer(ResidualBlock, 128, 2, stride=2)
    self.layer3 = self.make_layer(ResidualBlock, 256, 2, stride=2)
    self.layer4 = self.make_layer(ResidualBlock, 512, 2, stride=2)
    self.fc = nn.Linear(512, num_classes)

  def make_layer(self, block, channels, num_blocks, stride):
    strides = [stride] + [1] * (num_blocks - 1)   #strides=[1,1]
    layers = []
    for stride in strides:
        layers.append(block(self.inchannel, channels, stride))
        self.inchannel = channels
    return nn.Sequential(*layers)
  def forward(self, x):
    out = self.conv1(x)
    out = self.layer1(out)
    out = self.layer2(out)
    out = self.layer3(out)
    out = self.layer4(out)
    out = F.avg_pool2d(out, 4)
    out = out.view(out.size(0), -1)
    out = self.fc(out)
    return out


def ResNet18():
  return ResNet(ResidualBlock)

modelnew_ = ResNet18().to(device)  ###81.3
criterion = nn.CrossEntropyLoss()
```

```
#optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
optimizer = torch.optim.SGD(modelnew_.parameters(), lr = 0.001, momentum = 0.9, w
metrics_ = train(modelnew_, train_loader, test_loader, criterion, optimizer, 50)
```

What changes did you make to improve your model?

In the convolutional layer, added batch normalization and relu. Added four layers in residual blocks, each layer applied 2 convolutional layers, batch normalization and relu for the first convolutional layer. Updated stride from 1 to 2 in the last 3 residual blocks and applied average pooling.

In [0]:
```
plot_graphs("AwesomeModel", metrics)
```

After you get a nice model, download the test_file.zip and unzip it to get test_file.pt. In colab, you can explore your files from the left side bar. You can also download the files to your machine from there.

In [0]:
```
!wget http://courses.engr.illinois.edu/cs498aml/sp2019/homeworks/test_file.zip
!unzip test_file.zip
```

```
--2019-04-22 19:10:03--  http://courses.engr.illinois.edu/cs498aml/sp2019/homework
s/test_file.zip (http://courses.engr.illinois.edu/cs498aml/sp2019/homeworks/test_f
ile.zip)
Resolving courses.engr.illinois.edu (courses.engr.illinois.edu)... 130.126.151.9
Connecting to courses.engr.illinois.edu (courses.engr.illinois.edu)|130.126.151.9
|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://courses.engr.illinois.edu/cs498aml/sp2019/homeworks/test_file.zi
p (https://courses.engr.illinois.edu/cs498aml/sp2019/homeworks/test_file.zip) [fol
lowing]
--2019-04-22 19:10:04--  https://courses.engr.illinois.edu/cs498aml/sp2019/homewor
ks/test_file.zip (https://courses.engr.illinois.edu/cs498aml/sp2019/homeworks/test
_file.zip)
Connecting to courses.engr.illinois.edu (courses.engr.illinois.edu)|130.126.151.9
|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3841776 (3.7M) [application/x-zip-compressed]
Saving to: 'test_file.zip'

test_file.zip       100%[===================>]   3.66M  1.56MB/s    in 2.4s

2019-04-22 19:10:07 (1.56 MB/s) - 'test_file.zip' saved [3841776/3841776]

Archive:  test_file.zip
  inflating: test_file.pt
```

Then use your model to predict the label of the test images. Fill the remaining code below, where x has two dimensions (batch_size x one image size). Remember to reshpe x accordingly before feeding it into your model. The submission.txt should contain one predicted label (0~9) each line. Submit your submission.txt to the competition in gradscope.

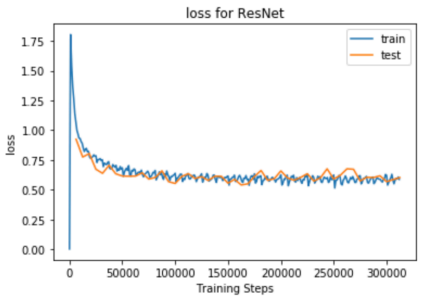
```
  Epoch 0 @ step 0: Train Loss: 0.004337, Train Accuracy: 0.000000
  Epoch 0 @ step 1000: Train Loss: 1.802889, Train Accuracy: 0.329375
  Epoch 0 @ step 2000: Train Loss: 1.540564, Train Accuracy: 0.444250
  Epoch 0 @ step 3000: Train Loss: 1.371203, Train Accuracy: 0.514375
  Epoch 0 @ step 4000: Train Loss: 1.266505, Train Accuracy: 0.549375
  Epoch 0 @ step 5000: Train Loss: 1.141689, Train Accuracy: 0.596250
  Epoch 0 @ step 6000: Train Loss: 1.069220, Train Accuracy: 0.630625
Epoch 0 Test Loss: 0.924513, Test Accuracy: 0.682900, time: 129.7s
  Epoch 1 @ step 7000: Train Loss: 1.000708, Train Accuracy: 0.656375
  Epoch 1 @ step 8000: Train Loss: 0.969487, Train Accuracy: 0.672125
  Epoch 1 @ step 9000: Train Loss: 0.933725, Train Accuracy: 0.686875
  Epoch 1 @ step 10000: Train Loss: 0.931493, Train Accuracy: 0.687500
  Epoch 1 @ step 11000: Train Loss: 0.905032, Train Accuracy: 0.697250
  Epoch 1 @ step 12000: Train Loss: 0.891164, Train Accuracy: 0.701250
Epoch 1 Test Loss: 0.775671, Test Accuracy: 0.737500, time: 131.3s
  Epoch 2 @ step 13000: Train Loss: 0.851592, Train Accuracy: 0.711750
  Epoch 2 @ step 14000: Train Loss: 0.821638, Train Accuracy: 0.730000
  Epoch 2 @ step 15000: Train Loss: 0.841045, Train Accuracy: 0.718375
  Epoch 2 @ step 16000: Train Loss: 0.848464, Train Accuracy: 0.716750
  Epoch 2 @ step 17000: Train Loss: 0.819805, Train Accuracy: 0.731250
  Epoch 2 @ step 18000: Train Loss: 0.823097, Train Accuracy: 0.728750
Epoch 2 Test Loss: 0.804308, Test Accuracy: 0.732700, time: 130.9s
  Epoch 3 @ step 19000: Train Loss: 0.776930, Train Accuracy: 0.747250
  Epoch 3 @ step 20000: Train Loss: 0.764710, Train Accuracy: 0.751125
  Epoch 3 @ step 21000: Train Loss: 0.772259, Train Accuracy: 0.750625
  Epoch 3 @ step 22000: Train Loss: 0.778772, Train Accuracy: 0.750625
  Epoch 3 @ step 23000: Train Loss: 0.793569, Train Accuracy: 0.740000
  Epoch 3 @ step 24000: Train Loss: 0.782233, Train Accuracy: 0.741125
Epoch 3 Test Loss: 0.673204, Test Accuracy: 0.778900, time: 131.0s
  Epoch 4 @ step 25000: Train Loss: 0.786287, Train Accuracy: 0.744750
  Epoch 4 @ step 26000: Train Loss: 0.729571, Train Accuracy: 0.762375
  Epoch 4 @ step 27000: Train Loss: 0.754891, Train Accuracy: 0.756875
  Epoch 4 @ step 28000: Train Loss: 0.753949, Train Accuracy: 0.756750
  Epoch 4 @ step 29000: Train Loss: 0.762443, Train Accuracy: 0.752375
  Epoch 4 @ step 30000: Train Loss: 0.745728, Train Accuracy: 0.762000
  Epoch 4 @ step 31000: Train Loss: 0.753371, Train Accuracy: 0.757625
Epoch 4 Test Loss: 0.639000, Test Accuracy: 0.793000, time: 131.7s
  Epoch 5 @ step 32000: Train Loss: 0.694868, Train Accuracy: 0.779750
  Epoch 5 @ step 33000: Train Loss: 0.725568, Train Accuracy: 0.765750
  Epoch 5 @ step 34000: Train Loss: 0.716358, Train Accuracy: 0.766500
  Epoch 5 @ step 35000: Train Loss: 0.712187, Train Accuracy: 0.774250
  Epoch 5 @ step 36000: Train Loss: 0.727948, Train Accuracy: 0.764625
  Epoch 5 @ step 37000: Train Loss: 0.737050, Train Accuracy: 0.762000
Epoch 5 Test Loss: 0.706809, Test Accuracy: 0.768800, time: 131.2s
  Epoch 6 @ step 38000: Train Loss: 0.681453, Train Accuracy: 0.781250
  Epoch 6 @ step 39000: Train Loss: 0.676037, Train Accuracy: 0.780250
  Epoch 6 @ step 40000: Train Loss: 0.678420, Train Accuracy: 0.784625
  Epoch 6 @ step 41000: Train Loss: 0.712252, Train Accuracy: 0.773000
  Epoch 6 @ step 42000: Train Loss: 0.700412, Train Accuracy: 0.774000
  Epoch 6 @ step 43000: Train Loss: 0.689764, Train Accuracy: 0.779625
Epoch 6 Test Loss: 0.635372, Test Accuracy: 0.788900, time: 131.1s
  Epoch 7 @ step 44000: Train Loss: 0.705850, Train Accuracy: 0.777250
  Epoch 7 @ step 45000: Train Loss: 0.657209, Train Accuracy: 0.790750
  Epoch 7 @ step 46000: Train Loss: 0.676879, Train Accuracy: 0.784750
  Epoch 7 @ step 47000: Train Loss: 0.664248, Train Accuracy: 0.785625
  Epoch 7 @ step 48000: Train Loss: 0.695722, Train Accuracy: 0.779500
  Epoch 7 @ step 49000: Train Loss: 0.673262, Train Accuracy: 0.785375
Epoch 7 Test Loss: 0.616282, Test Accuracy: 0.795000, time: 131.3s
  Epoch 8 @ step 50000: Train Loss: 0.684728, Train Accuracy: 0.783000
  Epoch 8 @ step 51000: Train Loss: 0.614166, Train Accuracy: 0.808750
  Epoch 8 @ step 52000: Train Loss: 0.649171, Train Accuracy: 0.796125
  Epoch 8 @ step 53000: Train Loss: 0.670122, Train Accuracy: 0.789125
  Epoch 8 @ step 54000: Train Loss: 0.682316, Train Accuracy: 0.780875
  Epoch 8 @ step 55000: Train Loss: 0.662826, Train Accuracy: 0.788500
  Epoch 8 @ step 56000: Train Loss: 0.703769, Train Accuracy: 0.773875
Epoch 8 Test Loss: 0.615023, Test Accuracy: 0.795800, time: 131.7s
  Epoch 9 @ step 57000: Train Loss: 0.625287, Train Accuracy: 0.805375
  Epoch 9 @ step 58000: Train Loss: 0.616536, Train Accuracy: 0.803875
  Epoch 9 @ step 59000: Train Loss: 0.664725, Train Accuracy: 0.785500
  Epoch 9 @ step 60000: Train Loss: 0.656896, Train Accuracy: 0.790250
  Epoch 9 @ step 61000: Train Loss: 0.676026, Train Accuracy: 0.784625
  Epoch 9 @ step 62000: Train Loss: 0.668732, Train Accuracy: 0.782750
Epoch 9 Test Loss: 0.616053, Test Accuracy: 0.793000, time: 130.7s
  Epoch 10 @ step 63000: Train Loss: 0.623416, Train Accuracy: 0.803875
  Epoch 10 @ step 64000: Train Loss: 0.631674, Train Accuracy: 0.798000
  Epoch 10 @ step 65000: Train Loss: 0.647473, Train Accuracy: 0.794750
  Epoch 10 @ step 66000: Train Loss: 0.648936, Train Accuracy: 0.792875
  Epoch 10 @ step 67000: Train Loss: 0.644404, Train Accuracy: 0.795125
  Epoch 10 @ step 68000: Train Loss: 0.665462, Train Accuracy: 0.785000
Epoch 10 Test Loss: 0.643210, Test Accuracy: 0.783200, time: 131.1s
  Epoch 11 @ step 69000: Train Loss: 0.629413, Train Accuracy: 0.796250
  Epoch 11 @ step 70000: Train Loss: 0.609102, Train Accuracy: 0.807875
  Epoch 11 @ step 71000: Train Loss: 0.630306, Train Accuracy: 0.802875
  Epoch 11 @ step 72000: Train Loss: 0.636662, Train Accuracy: 0.799500
  Epoch 11 @ step 73000: Train Loss: 0.647417, Train Accuracy: 0.794875
  Epoch 11 @ step 74000: Train Loss: 0.657259, Train Accuracy: 0.792625
Epoch 11 Test Loss: 0.590246, Test Accuracy: 0.803300, time: 131.6s
  Epoch 12 @ step 75000: Train Loss: 0.639885, Train Accuracy: 0.795750
  Epoch 12 @ step 76000: Train Loss: 0.602334, Train Accuracy: 0.807875
  Epoch 12 @ step 77000: Train Loss: 0.610766, Train Accuracy: 0.807500
  Epoch 12 @ step 78000: Train Loss: 0.623604, Train Accuracy: 0.802625
  Epoch 12 @ step 79000: Train Loss: 0.646862, Train Accuracy: 0.796625
  Epoch 12 @ step 80000: Train Loss: 0.642317, Train Accuracy: 0.796625
  Epoch 12 @ step 81000: Train Loss: 0.662695, Train Accuracy: 0.786750
```

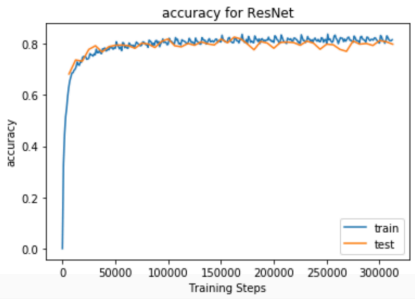
```
Epoch 12 Test Loss: 0.604526, Test Accuracy: 0.799600, time: 130.4s
  Epoch 13 @ step 82000: Train Loss: 0.583708, Train Accuracy: 0.815000
  Epoch 13 @ step 83000: Train Loss: 0.602148, Train Accuracy: 0.811625
  Epoch 13 @ step 84000: Train Loss: 0.622789, Train Accuracy: 0.807500
  Epoch 13 @ step 85000: Train Loss: 0.651079, Train Accuracy: 0.792500
  Epoch 13 @ step 86000: Train Loss: 0.628016, Train Accuracy: 0.803375
  Epoch 13 @ step 87000: Train Loss: 0.646626, Train Accuracy: 0.795625
Epoch 13 Test Loss: 0.660061, Test Accuracy: 0.786100, time: 131.4s
  Epoch 14 @ step 88000: Train Loss: 0.605481, Train Accuracy: 0.811750
  Epoch 14 @ step 89000: Train Loss: 0.588039, Train Accuracy: 0.813875
  Epoch 14 @ step 90000: Train Loss: 0.634131, Train Accuracy: 0.802125
  Epoch 14 @ step 91000: Train Loss: 0.611442, Train Accuracy: 0.809875
  Epoch 14 @ step 92000: Train Loss: 0.658429, Train Accuracy: 0.789375
  Epoch 14 @ step 93000: Train Loss: 0.628128, Train Accuracy: 0.801625
Epoch 14 Test Loss: 0.568931, Test Accuracy: 0.810000, time: 130.7s
  Epoch 15 @ step 94000: Train Loss: 0.610361, Train Accuracy: 0.807125
  Epoch 15 @ step 95000: Train Loss: 0.578164, Train Accuracy: 0.819750
  Epoch 15 @ step 96000: Train Loss: 0.612706, Train Accuracy: 0.805500
  Epoch 15 @ step 97000: Train Loss: 0.610430, Train Accuracy: 0.807500
  Epoch 15 @ step 98000: Train Loss: 0.622119, Train Accuracy: 0.806750
  Epoch 15 @ step 99000: Train Loss: 0.626306, Train Accuracy: 0.797125
Epoch 15 Test Loss: 0.553789, Test Accuracy: 0.819800, time: 130.4s
  Epoch 16 @ step 100000: Train Loss: 0.639551, Train Accuracy: 0.798500
  Epoch 16 @ step 101000: Train Loss: 0.569288, Train Accuracy: 0.822250
  Epoch 16 @ step 102000: Train Loss: 0.597008, Train Accuracy: 0.812250
  Epoch 16 @ step 103000: Train Loss: 0.604731, Train Accuracy: 0.811000
  Epoch 16 @ step 104000: Train Loss: 0.620558, Train Accuracy: 0.806375
  Epoch 16 @ step 105000: Train Loss: 0.632557, Train Accuracy: 0.802500
  Epoch 16 @ step 106000: Train Loss: 0.630764, Train Accuracy: 0.800375
Epoch 16 Test Loss: 0.613105, Test Accuracy: 0.793000, time: 130.8s
  Epoch 17 @ step 107000: Train Loss: 0.570313, Train Accuracy: 0.824875
  Epoch 17 @ step 108000: Train Loss: 0.593737, Train Accuracy: 0.814875
  Epoch 17 @ step 109000: Train Loss: 0.587276, Train Accuracy: 0.818500
  Epoch 17 @ step 110000: Train Loss: 0.618949, Train Accuracy: 0.800625
  Epoch 17 @ step 111000: Train Loss: 0.623250, Train Accuracy: 0.803250
  Epoch 17 @ step 112000: Train Loss: 0.637129, Train Accuracy: 0.796500
Epoch 17 Test Loss: 0.636240, Test Accuracy: 0.788900, time: 131.0s
  Epoch 18 @ step 113000: Train Loss: 0.597065, Train Accuracy: 0.816625
  Epoch 18 @ step 114000: Train Loss: 0.590398, Train Accuracy: 0.812875
  Epoch 18 @ step 115000: Train Loss: 0.600803, Train Accuracy: 0.811375
  Epoch 18 @ step 116000: Train Loss: 0.619374, Train Accuracy: 0.809125
  Epoch 18 @ step 117000: Train Loss: 0.617611, Train Accuracy: 0.804000
  Epoch 18 @ step 118000: Train Loss: 0.625002, Train Accuracy: 0.800625
Epoch 18 Test Loss: 0.595384, Test Accuracy: 0.802000, time: 130.3s
  Epoch 19 @ step 119000: Train Loss: 0.610516, Train Accuracy: 0.806500
  Epoch 19 @ step 120000: Train Loss: 0.569148, Train Accuracy: 0.822125
  Epoch 19 @ step 121000: Train Loss: 0.591577, Train Accuracy: 0.815000
  Epoch 19 @ step 122000: Train Loss: 0.606244, Train Accuracy: 0.805375
  Epoch 19 @ step 123000: Train Loss: 0.610974, Train Accuracy: 0.808500
  Epoch 19 @ step 124000: Train Loss: 0.618476, Train Accuracy: 0.803750
Epoch 19 Test Loss: 0.610895, Test Accuracy: 0.794800, time: 130.8s
  Epoch 20 @ step 125000: Train Loss: 0.635362, Train Accuracy: 0.798500
  Epoch 20 @ step 126000: Train Loss: 0.564972, Train Accuracy: 0.823250
  Epoch 20 @ step 127000: Train Loss: 0.593074, Train Accuracy: 0.812375
  Epoch 20 @ step 128000: Train Loss: 0.608094, Train Accuracy: 0.808625
  Epoch 20 @ step 129000: Train Loss: 0.602284, Train Accuracy: 0.806125
  Epoch 20 @ step 130000: Train Loss: 0.637150, Train Accuracy: 0.799750
  Epoch 20 @ step 131000: Train Loss: 0.623780, Train Accuracy: 0.799250
Epoch 20 Test Loss: 0.578735, Test Accuracy: 0.808300, time: 131.2s
  Epoch 21 @ step 132000: Train Loss: 0.573708, Train Accuracy: 0.824750
  Epoch 21 @ step 133000: Train Loss: 0.580969, Train Accuracy: 0.818750
  Epoch 21 @ step 134000: Train Loss: 0.588649, Train Accuracy: 0.814000
  Epoch 21 @ step 135000: Train Loss: 0.598084, Train Accuracy: 0.814375
  Epoch 21 @ step 136000: Train Loss: 0.623672, Train Accuracy: 0.799500
  Epoch 21 @ step 137000: Train Loss: 0.623747, Train Accuracy: 0.804125
Epoch 21 Test Loss: 0.615231, Test Accuracy: 0.800300, time: 130.7s
  Epoch 22 @ step 138000: Train Loss: 0.587075, Train Accuracy: 0.815250
  Epoch 22 @ step 139000: Train Loss: 0.562753, Train Accuracy: 0.820875
  Epoch 22 @ step 140000: Train Loss: 0.611376, Train Accuracy: 0.808500
  Epoch 22 @ step 141000: Train Loss: 0.613087, Train Accuracy: 0.809625
  Epoch 22 @ step 142000: Train Loss: 0.602276, Train Accuracy: 0.812875
  Epoch 22 @ step 143000: Train Loss: 0.610165, Train Accuracy: 0.807250
Epoch 22 Test Loss: 0.613477, Test Accuracy: 0.796000, time: 131.6s
  Epoch 23 @ step 144000: Train Loss: 0.602597, Train Accuracy: 0.812250
  Epoch 23 @ step 145000: Train Loss: 0.577962, Train Accuracy: 0.819125
  Epoch 23 @ step 146000: Train Loss: 0.585098, Train Accuracy: 0.818500
  Epoch 23 @ step 147000: Train Loss: 0.605537, Train Accuracy: 0.806375
  Epoch 23 @ step 148000: Train Loss: 0.601281, Train Accuracy: 0.813500
  Epoch 23 @ step 149000: Train Loss: 0.611494, Train Accuracy: 0.810125
Epoch 23 Test Loss: 0.556663, Test Accuracy: 0.819400, time: 131.3s
  Epoch 24 @ step 150000: Train Loss: 0.616815, Train Accuracy: 0.807750
  Epoch 24 @ step 151000: Train Loss: 0.539774, Train Accuracy: 0.834250
  Epoch 24 @ step 152000: Train Loss: 0.590327, Train Accuracy: 0.817500
  Epoch 24 @ step 153000: Train Loss: 0.591353, Train Accuracy: 0.814250
  Epoch 24 @ step 154000: Train Loss: 0.604631, Train Accuracy: 0.808125
  Epoch 24 @ step 155000: Train Loss: 0.606467, Train Accuracy: 0.807875
  Epoch 24 @ step 156000: Train Loss: 0.623726, Train Accuracy: 0.808500
Epoch 24 Test Loss: 0.588057, Test Accuracy: 0.804600, time: 131.2s
  Epoch 25 @ step 157000: Train Loss: 0.571643, Train Accuracy: 0.823125
  Epoch 25 @ step 158000: Train Loss: 0.571936, Train Accuracy: 0.825500
  Epoch 25 @ step 159000: Train Loss: 0.587229, Train Accuracy: 0.811875
  Epoch 25 @ step 160000: Train Loss: 0.611392, Train Accuracy: 0.807875
  Epoch 25 @ step 161000: Train Loss: 0.622406, Train Accuracy: 0.804500
  Epoch 25 @ step 162000: Train Loss: 0.601554, Train Accuracy: 0.805500
Epoch 25 Test Loss: 0.541391, Test Accuracy: 0.827000, time: 131.7s
  Epoch 26 @ step 163000: Train Loss: 0.572380, Train Accuracy: 0.822750
  Epoch 26 @ step 164000: Train Loss: 0.561954, Train Accuracy: 0.823500
  Epoch 26 @ step 165000: Train Loss: 0.598593, Train Accuracy: 0.812125
  Epoch 26 @ step 166000: Train Loss: 0.595584, Train Accuracy: 0.812250
```

```
  Epoch 26 @ step 167000: Train Loss: 0.619989, Train Accuracy: 0.804000
  Epoch 26 @ step 168000: Train Loss: 0.587452, Train Accuracy: 0.817125
Epoch 26 Test Loss: 0.550927, Test Accuracy: 0.819600, time: 131.4s
  Epoch 27 @ step 169000: Train Loss: 0.589197, Train Accuracy: 0.816875
  Epoch 27 @ step 170000: Train Loss: 0.538183, Train Accuracy: 0.837875
  Epoch 27 @ step 171000: Train Loss: 0.598497, Train Accuracy: 0.816625
  Epoch 27 @ step 172000: Train Loss: 0.617069, Train Accuracy: 0.807000
  Epoch 27 @ step 173000: Train Loss: 0.618135, Train Accuracy: 0.806750
  Epoch 27 @ step 174000: Train Loss: 0.586697, Train Accuracy: 0.817625
Epoch 27 Test Loss: 0.612253, Test Accuracy: 0.800100, time: 131.9s
  Epoch 28 @ step 175000: Train Loss: 0.635343, Train Accuracy: 0.800000
  Epoch 28 @ step 176000: Train Loss: 0.557162, Train Accuracy: 0.829750
  Epoch 28 @ step 177000: Train Loss: 0.570805, Train Accuracy: 0.820500
  Epoch 28 @ step 178000: Train Loss: 0.613255, Train Accuracy: 0.806375
  Epoch 28 @ step 179000: Train Loss: 0.603208, Train Accuracy: 0.812375
  Epoch 28 @ step 180000: Train Loss: 0.614475, Train Accuracy: 0.806625
  Epoch 28 @ step 181000: Train Loss: 0.618972, Train Accuracy: 0.804375
Epoch 28 Test Loss: 0.662619, Test Accuracy: 0.777300, time: 131.2s
  Epoch 29 @ step 182000: Train Loss: 0.543544, Train Accuracy: 0.831375
  Epoch 29 @ step 183000: Train Loss: 0.576047, Train Accuracy: 0.820250
  Epoch 29 @ step 184000: Train Loss: 0.582041, Train Accuracy: 0.817250
  Epoch 29 @ step 185000: Train Loss: 0.602556, Train Accuracy: 0.809500
  Epoch 29 @ step 186000: Train Loss: 0.590241, Train Accuracy: 0.819875
  Epoch 29 @ step 187000: Train Loss: 0.615872, Train Accuracy: 0.802125
Epoch 29 Test Loss: 0.579806, Test Accuracy: 0.809500, time: 132.3s
  Epoch 30 @ step 188000: Train Loss: 0.589801, Train Accuracy: 0.818375
  Epoch 30 @ step 189000: Train Loss: 0.573001, Train Accuracy: 0.820875
  Epoch 30 @ step 190000: Train Loss: 0.585080, Train Accuracy: 0.813125
  Epoch 30 @ step 191000: Train Loss: 0.595990, Train Accuracy: 0.814750
  Epoch 30 @ step 192000: Train Loss: 0.606288, Train Accuracy: 0.811000
  Epoch 30 @ step 193000: Train Loss: 0.591226, Train Accuracy: 0.816250
Epoch 30 Test Loss: 0.593658, Test Accuracy: 0.803600, time: 131.9s
  Epoch 31 @ step 194000: Train Loss: 0.587176, Train Accuracy: 0.815500
  Epoch 31 @ step 195000: Train Loss: 0.565677, Train Accuracy: 0.826625
  Epoch 31 @ step 196000: Train Loss: 0.585854, Train Accuracy: 0.815375
  Epoch 31 @ step 197000: Train Loss: 0.590277, Train Accuracy: 0.815625
  Epoch 31 @ step 198000: Train Loss: 0.596259, Train Accuracy: 0.809125
  Epoch 31 @ step 199000: Train Loss: 0.605485, Train Accuracy: 0.813000
Epoch 31 Test Loss: 0.659187, Test Accuracy: 0.783300, time: 131.0s
  Epoch 32 @ step 200000: Train Loss: 0.626464, Train Accuracy: 0.805625
  Epoch 32 @ step 201000: Train Loss: 0.540875, Train Accuracy: 0.832250
  Epoch 32 @ step 202000: Train Loss: 0.558553, Train Accuracy: 0.827500
  Epoch 32 @ step 203000: Train Loss: 0.592086, Train Accuracy: 0.817250
  Epoch 32 @ step 204000: Train Loss: 0.586808, Train Accuracy: 0.814625
  Epoch 32 @ step 205000: Train Loss: 0.614386, Train Accuracy: 0.802750
  Epoch 32 @ step 206000: Train Loss: 0.623473, Train Accuracy: 0.804250
Epoch 32 Test Loss: 0.600519, Test Accuracy: 0.806100, time: 131.8s
  Epoch 33 @ step 207000: Train Loss: 0.534909, Train Accuracy: 0.835375
  Epoch 33 @ step 208000: Train Loss: 0.565287, Train Accuracy: 0.825000
  Epoch 33 @ step 209000: Train Loss: 0.591958, Train Accuracy: 0.813000
  Epoch 33 @ step 210000: Train Loss: 0.614159, Train Accuracy: 0.807875
  Epoch 33 @ step 211000: Train Loss: 0.604547, Train Accuracy: 0.806750
  Epoch 33 @ step 212000: Train Loss: 0.623032, Train Accuracy: 0.801750
  Epoch 33 @ step 213000: Train Loss: 0.611753, Train Accuracy: 0.808875
Epoch 35 Test Loss: 0.635686, Test Accuracy: 0.794900, time: 131.3s
  Epoch 36 @ step 225000: Train Loss: 0.630844, Train Accuracy: 0.801000
  Epoch 36 @ step 226000: Train Loss: 0.534990, Train Accuracy: 0.835875
  Epoch 36 @ step 227000: Train Loss: 0.564182, Train Accuracy: 0.824625
  Epoch 36 @ step 228000: Train Loss: 0.598609, Train Accuracy: 0.813000
  Epoch 36 @ step 229000: Train Loss: 0.592798, Train Accuracy: 0.813375
  Epoch 36 @ step 230000: Train Loss: 0.618922, Train Accuracy: 0.805625
  Epoch 36 @ step 231000: Train Loss: 0.606268, Train Accuracy: 0.811625
Epoch 36 Test Loss: 0.573954, Test Accuracy: 0.811200, time: 131.8s
  Epoch 37 @ step 232000: Train Loss: 0.557094, Train Accuracy: 0.829375
  Epoch 37 @ step 233000: Train Loss: 0.575041, Train Accuracy: 0.819000
  Epoch 37 @ step 234000: Train Loss: 0.583873, Train Accuracy: 0.818500
  Epoch 37 @ step 235000: Train Loss: 0.615000, Train Accuracy: 0.808000
  Epoch 37 @ step 236000: Train Loss: 0.595324, Train Accuracy: 0.816500
  Epoch 37 @ step 237000: Train Loss: 0.610656, Train Accuracy: 0.807375
Epoch 37 Test Loss: 0.602274, Test Accuracy: 0.803300, time: 130.5s
  Epoch 38 @ step 238000: Train Loss: 0.569025, Train Accuracy: 0.818000
  Epoch 38 @ step 239000: Train Loss: 0.561798, Train Accuracy: 0.823250
  Epoch 38 @ step 240000: Train Loss: 0.587188, Train Accuracy: 0.815125
  Epoch 38 @ step 241000: Train Loss: 0.608030, Train Accuracy: 0.810625
  Epoch 38 @ step 242000: Train Loss: 0.597632, Train Accuracy: 0.818000
  Epoch 38 @ step 243000: Train Loss: 0.594378, Train Accuracy: 0.809625
Epoch 38 Test Loss: 0.676789, Test Accuracy: 0.779400, time: 131.2s
  Epoch 39 @ step 244000: Train Loss: 0.603703, Train Accuracy: 0.808500
  Epoch 39 @ step 245000: Train Loss: 0.562063, Train Accuracy: 0.824625
  Epoch 39 @ step 246000: Train Loss: 0.571067, Train Accuracy: 0.822125
  Epoch 39 @ step 247000: Train Loss: 0.610644, Train Accuracy: 0.806625
  Epoch 39 @ step 248000: Train Loss: 0.586551, Train Accuracy: 0.816875
  Epoch 39 @ step 249000: Train Loss: 0.588023, Train Accuracy: 0.817250
Epoch 39 Test Loss: 0.590779, Test Accuracy: 0.799300, time: 131.0s
  Epoch 40 @ step 250000: Train Loss: 0.624519, Train Accuracy: 0.802375
  Epoch 40 @ step 251000: Train Loss: 0.515699, Train Accuracy: 0.838500
```

```
  Epoch 40 @ step 252000: Train Loss: 0.580088, Train Accuracy: 0.815625
  Epoch 40 @ step 253000: Train Loss: 0.581467, Train Accuracy: 0.820375
  Epoch 40 @ step 254000: Train Loss: 0.601320, Train Accuracy: 0.812250
  Epoch 40 @ step 255000: Train Loss: 0.621717, Train Accuracy: 0.805875
  Epoch 40 @ step 256000: Train Loss: 0.608692, Train Accuracy: 0.810125
Epoch 40 Test Loss: 0.622476, Test Accuracy: 0.796900, time: 130.4s
  Epoch 41 @ step 257000: Train Loss: 0.549310, Train Accuracy: 0.830125
  Epoch 41 @ step 258000: Train Loss: 0.541160, Train Accuracy: 0.834625
  Epoch 41 @ step 259000: Train Loss: 0.581388, Train Accuracy: 0.820875
  Epoch 41 @ step 260000: Train Loss: 0.593374, Train Accuracy: 0.816375
  Epoch 41 @ step 261000: Train Loss: 0.617721, Train Accuracy: 0.801250
  Epoch 41 @ step 262000: Train Loss: 0.598570, Train Accuracy: 0.812875
Epoch 41 Test Loss: 0.676593, Test Accuracy: 0.778900, time: 131.4s
  Epoch 42 @ step 263000: Train Loss: 0.565906, Train Accuracy: 0.824875
  Epoch 42 @ step 264000: Train Loss: 0.558389, Train Accuracy: 0.822625
  Epoch 42 @ step 265000: Train Loss: 0.581225, Train Accuracy: 0.822000
  Epoch 42 @ step 266000: Train Loss: 0.604199, Train Accuracy: 0.809250
  Epoch 42 @ step 267000: Train Loss: 0.589154, Train Accuracy: 0.818750
  Epoch 42 @ step 268000: Train Loss: 0.612255, Train Accuracy: 0.810625
Epoch 42 Test Loss: 0.672803, Test Accuracy: 0.771200, time: 131.4s
  Epoch 43 @ step 269000: Train Loss: 0.591223, Train Accuracy: 0.817250
  Epoch 43 @ step 270000: Train Loss: 0.562753, Train Accuracy: 0.825375
  Epoch 43 @ step 271000: Train Loss: 0.562580, Train Accuracy: 0.826375
  Epoch 43 @ step 272000: Train Loss: 0.590341, Train Accuracy: 0.814250
  Epoch 43 @ step 273000: Train Loss: 0.596985, Train Accuracy: 0.814875
  Epoch 43 @ step 274000: Train Loss: 0.617158, Train Accuracy: 0.809875
Epoch 43 Test Loss: 0.574543, Test Accuracy: 0.811700, time: 131.3s
  Epoch 44 @ step 275000: Train Loss: 0.618908, Train Accuracy: 0.806250
  Epoch 44 @ step 276000: Train Loss: 0.550251, Train Accuracy: 0.830875
  Epoch 44 @ step 277000: Train Loss: 0.567811, Train Accuracy: 0.824500
  Epoch 44 @ step 278000: Train Loss: 0.594829, Train Accuracy: 0.813750
  Epoch 44 @ step 279000: Train Loss: 0.596796, Train Accuracy: 0.812500
  Epoch 44 @ step 280000: Train Loss: 0.605438, Train Accuracy: 0.811750
  Epoch 44 @ step 281000: Train Loss: 0.613613, Train Accuracy: 0.803625
Epoch 44 Test Loss: 0.605824, Test Accuracy: 0.799200, time: 131.4s
  Epoch 45 @ step 282000: Train Loss: 0.561717, Train Accuracy: 0.827125
  Epoch 45 @ step 283000: Train Loss: 0.556910, Train Accuracy: 0.829875
  Epoch 45 @ step 284000: Train Loss: 0.575601, Train Accuracy: 0.819250
  Epoch 45 @ step 285000: Train Loss: 0.595951, Train Accuracy: 0.817000
  Epoch 45 @ step 286000: Train Loss: 0.599050, Train Accuracy: 0.813375
  Epoch 45 @ step 287000: Train Loss: 0.607438, Train Accuracy: 0.807500
Epoch 45 Test Loss: 0.603232, Test Accuracy: 0.801900, time: 131.2s
  Epoch 46 @ step 288000: Train Loss: 0.552328, Train Accuracy: 0.826750
  Epoch 46 @ step 289000: Train Loss: 0.563633, Train Accuracy: 0.825875
  Epoch 46 @ step 290000: Train Loss: 0.586499, Train Accuracy: 0.817750
  Epoch 46 @ step 291000: Train Loss: 0.602477, Train Accuracy: 0.812875
  Epoch 46 @ step 292000: Train Loss: 0.593274, Train Accuracy: 0.818000
  Epoch 46 @ step 293000: Train Loss: 0.589585, Train Accuracy: 0.818000
Epoch 46 Test Loss: 0.618490, Test Accuracy: 0.794000, time: 131.6s
  Epoch 47 @ step 294000: Train Loss: 0.588890, Train Accuracy: 0.814750
  Epoch 47 @ step 295000: Train Loss: 0.575366, Train Accuracy: 0.822750
  Epoch 47 @ step 296000: Train Loss: 0.565359, Train Accuracy: 0.824250
  Epoch 47 @ step 297000: Train Loss: 0.585192, Train Accuracy: 0.816750
  Epoch 47 @ step 298000: Train Loss: 0.594573, Train Accuracy: 0.812500
  Epoch 47 @ step 299000: Train Loss: 0.629340, Train Accuracy: 0.806750
Epoch 47 Test Loss: 0.568990, Test Accuracy: 0.813400, time: 131.5s
  Epoch 48 @ step 300000: Train Loss: 0.622180, Train Accuracy: 0.803375
  Epoch 48 @ step 301000: Train Loss: 0.540087, Train Accuracy: 0.833250
  Epoch 48 @ step 302000: Train Loss: 0.562745, Train Accuracy: 0.826750
  Epoch 48 @ step 303000: Train Loss: 0.584538, Train Accuracy: 0.814125
  Epoch 48 @ step 304000: Train Loss: 0.599627, Train Accuracy: 0.812250
  Epoch 48 @ step 305000: Train Loss: 0.630748, Train Accuracy: 0.802750
  Epoch 48 @ step 306000: Train Loss: 0.584656, Train Accuracy: 0.812250
Epoch 48 Test Loss: 0.586498, Test Accuracy: 0.808900, time: 131.7s
  Epoch 49 @ step 307000: Train Loss: 0.550965, Train Accuracy: 0.831875
  Epoch 49 @ step 308000: Train Loss: 0.580784, Train Accuracy: 0.824375
  Epoch 49 @ step 309000: Train Loss: 0.595933, Train Accuracy: 0.815375
  Epoch 49 @ step 310000: Train Loss: 0.590633, Train Accuracy: 0.812750
  Epoch 49 @ step 311000: Train Loss: 0.607528, Train Accuracy: 0.812500
  Epoch 49 @ step 312000: Train Loss: 0.588453, Train Accuracy: 0.816750
Epoch 49 Test Loss: 0.603380, Test Accuracy: 0.799100, time: 131.1s
Final Test Loss: 0.603380, Test Accuracy: 0.799100, Total time: 6560.4s
```

```
plot_graphs("ResNet", metrics_)
```





In [0]:

```
import torch.utils.data as Data

test_file = 'test_file.pt'
pred_file = 'submission.txt'

f_pred = open(pred_file,'w')
tensor = torch.load(test_file)
torch_dataset = Data.TensorDataset(tensor)
test_loader2 = torch.utils.data.DataLoader(torch_dataset, batch_size, shuffle=Fal

for ele in test_loader2:
    x = ele[0]

    # Fill your code here
    x = x.reshape(batch_size,3,32,32)
    predict = model(x.to(device))
    labels = torch.argmax(predict, dim = 1)
    for i in labels:
      f_pred.write(str(i.item()))
      f_pred.write('\n')

    #f_pred.write('\n')

f_pred.close()
```

Report

Part 0: Imports and Basic Setup (5 Points)

Nothing to report for this part. You will be just scored for finishing the setup.

Part 1: Fully connected neural networks (25 Points)

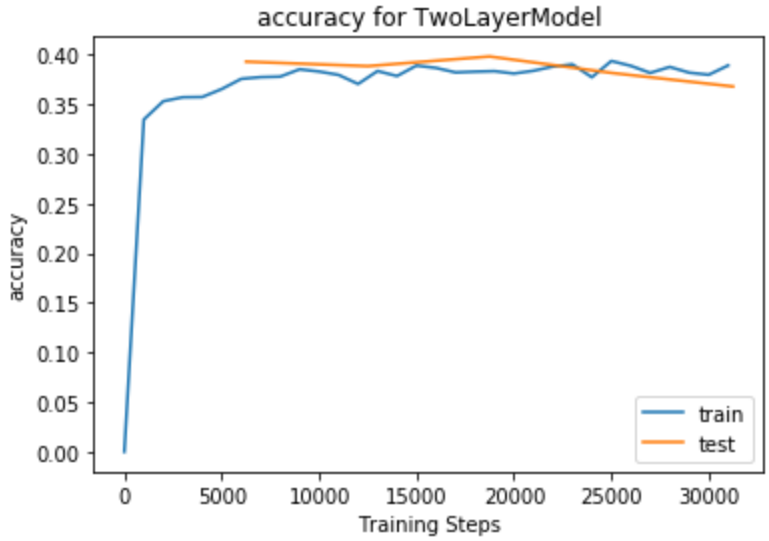
Test (on validation set) accuracy (5 Points): 0.367700

Test loss (5 Points): 1.750870

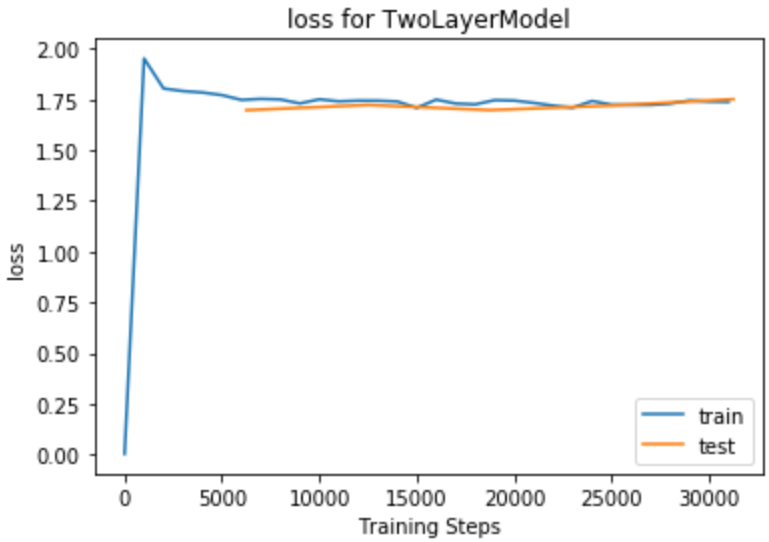
Training time (5 Points): 88.5s

Plots:

- Plot a graph of accuracy on validation set vs epoch (5 Points)



- Plot a graph of loss on validation set vs epoch (5 Points)



Part 2: Convolution Network (Basic) (35 Points)

Tensor dimensions: A good way to debug your network for size mismatches is to print the dimension of output after every layers:

(10 Points)

Output dimension after 1st conv layer: Size([8, 16, 32, 32])

Output dimension after 1st max pooling: Size([8, 16, 16, 16])

Output dimension after 2nd conv layer: Size([8, 16, 16, 16])

Output dimension after 2nd max pooling: No 2nd max pooling

Output dimension after 1st fully connected layer: 16x16x16, 64

Output dimension after 2nd fully connected layer: 64, 10

Output dimension after 3rd fully connected layer: No 3rd fully connected layer

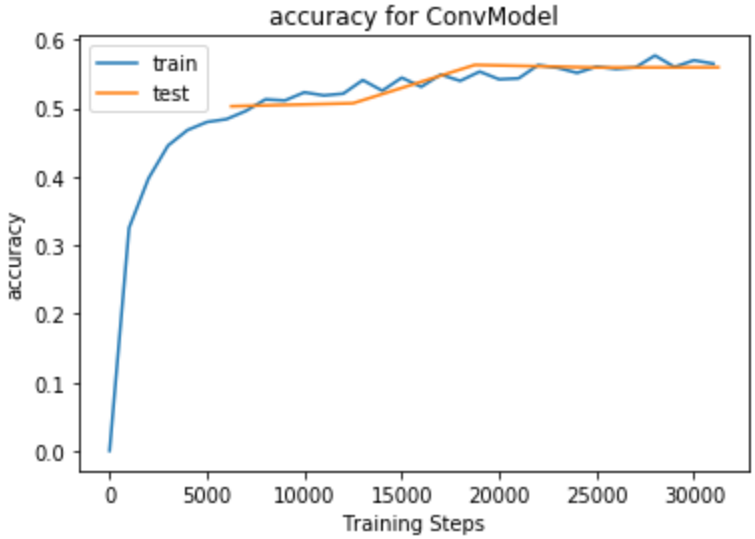
Test (on validation set) Accuracy (5 Points): 0.559500

Test loss (5 Points): 1.209426

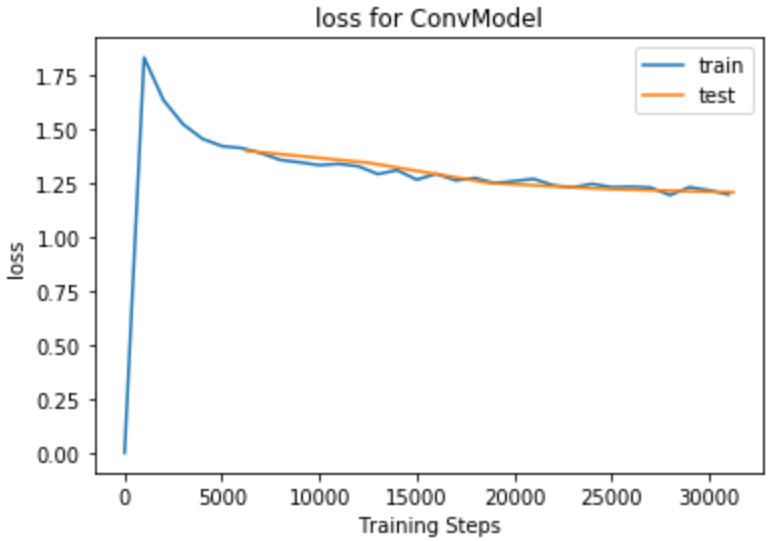
Training time (5 Points): 127.7s

Plots:

- Plot a graph of accuracy on validation set vs epoch (5 Points)



- Plot a graph of loss on validation set vs epoch (5 Points)



Part 3: Convolution Network (Add one or more suggested changes) (35 Points)

Describe the additional changes implemented, your intuition for as to why it works, you may also describe other approaches you experimented with (10 Points):

I applied Residual Neural Networks for this part to get a higher accuracy score by solving the vanishing gradient issue. By trying other models, I found out that the accuray score didn't imporve much by applying drop out, which means it may not due to the overfitting issue, but the vanishing gradients. For each layer in the residual blocks, there are two convolutional layers and batch normalization applied in both layers. Relu applied only in the first convolutional layer. Inside the residual block, layer learned from itself and its residual.The final model starts with 1 convolutional layers, batch normalization and relu then 4 residual block layers, and applied average pooling, flatten and final fully connected linear layer. In the last 3 residual layers, updated stride from 1 to 2 to decrease the image size. Also, I trained longer for 50 epochs (may get higher accurcy score by increasing the epochs number) and updated the optimizer method from Adam to SGD with momentum. It seems that by applying the SGD optimizer, residual neural network could get higher accuray than Adam optimizer.

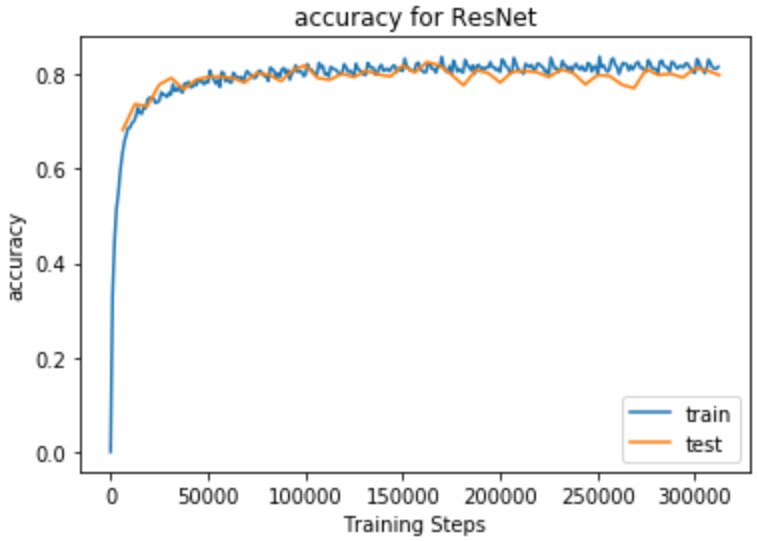
Test (on validation set) Accuracy (5 Points): 0.799100

Test loss (5 Points): 0.603380

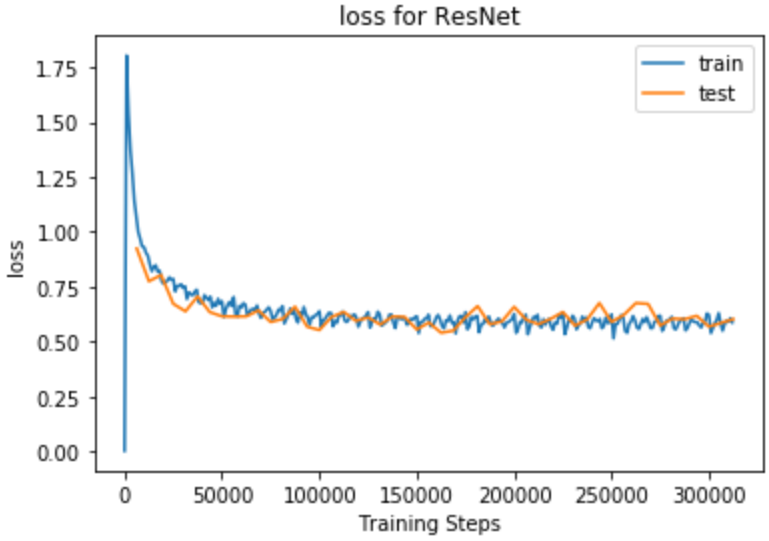
Training time (5 Points): 6560.4s

Plots:

- Plot a graph of accuracy on validation set vs epoch (5 Points)



- Plot a graph of loss on validation set vs epoch (5 Points)



10 bonus points will be awarded to top 3 scorers on leaderboard (in case of tie for 3rd position everyone tied for 3rd position will get the bonus)